誰にでも書ける #! /bin/sed -f 講座
第 1 回
「逆順のフィルタ」

安岡孝一

yasuoka : root さん、 root さん。
root : 何だい？
yasuoka : takahash くんから、 tacっていうコマンドをもらったんですけど

```
/home/yasuoka/bin% cat tac （ぽこ）
#! /bin/sed -f
1!G
h
$p
d
/home/yasuoka/bin% 
```

何だかわけのわかんない字が並んでるんです。
root : どういうコマンドなんだい？
yasuoka : 行の順序を逆にしてくれます。例えば

```
/home/yasuoka/bin% who （ぽこ）
yasuoka  console Mar  9 15:27
ochi     ttyp0   Mar  9 10:24   (daikaku)
takahash ttyp2   Mar  9 09:21   (kinkaku:0.0)
/home/yasuoka/bin% who | tac （ぽこ）
takahash ttyp2   Mar  9 09:21   (kinkaku:0.0)
ochi     ttyp0   Mar  9 10:24   (daikaku)
yasuoka  console Mar  9 15:27
/home/yasuoka/bin% 
```

とかいう風に、標準入力の行の順序を逆転することもできるし

```
/home/yasuoka/bin% tac tac （ぽこ）
d
$p
h
1!G
#! /bin/sed -f
/home/yasuoka/bin% 
```

てな風に、ファイルの行を逆順に出力させることもできます。
root : ははーん。で、どこがわからないの？
yasuoka : 全部です。

```
/home/yasuoka/bin% cat tac （ぽこ）
#! /bin/sed -f
1!G
h
$p
d
/home/yasuoka/bin% 
```

root : そうか、じゃまずは sed の基本から話さなきゃならないな。
yasuoka : sed？
root : うん、 sed はフィルタを書くためのインタプリタ型言語なんだ。
yasuoka : フィルタって何ですか？
root : この tac みたいに、標準入力から入力して標準出力に出力するタイプのコマンドだよ。入力はファイルでもいいけどね。まあ、 cat に少し毛のはえたようなコマンドかな。
yasuoka : はあ。
root : 1 行目の #! /bin/sed -fってのが、 sed のスクリプトだっていう宣言みたいなもんだ。ただし、これは BSD に限ったことで、 System V では最初の行には exec sed "‘sed 1d $0‘" ${1+"$@"} と書く。
yasuoka : えらく長いんですね。
root : まあね。で、 2 行目から最後までは、 1 行に 1 つずつ sed の命令が書いてある。
yasuoka : この 1!G とか d とかが、 sed の命令なんですか？
root : そう。 sed のスクリプトでは、 Unix のコマンドは全然使えないんだ。使えるのは sed の命令だけ。
yasuoka : そうなんですか。じゃ、その sed の命令ってのを教えて下さい。
root : うん、まず覚えておかなきゃいけないのは、 sed の命令は基本的に 1 文字だってこと。

yasuoka ： はい。

root ： sed の処理はバッファを介しておこなわれるってこと。このバッファってのは、入力を蓄えておくための内部的な場所だと思えばいい。

yasuoka ： はい。

root ： それからバッファの内容を一時退避しておくための、コピーバッファってのもある。 sed のスクリプトが始動した時には、バッファには入力の第 1 行目が入ってるし、コピーバッファには何が入ってるかわからない。

yasuoka ： はい、わかりました。

root ： あと if の書き方だけど、 sed ではどの命令でも、直前に実行条件を書くことができるんだ。

数	バッファに読み込まれているのが入力の「数」行目だったなら真
$	バッファに読み込まれているのが入力の最終行だったなら真
/正規表現/	バッファが正規表現とマッチしたなら真
条件,条件	左条件が真になってから右条件が真になるまでの間、真
条件!	条件が真でないとき真

yasuoka ： すると tac の 2 行目の 1!Gってのは、入力の 1 行目以外では G を実行するってことですか？

root ： よくわかったね。

yasuoka ： Gって何ですか？

root ： その前に 3 行目の h を教えておこう。

h
　バッファの内容を、コピーバッファに写す。
H
　コピーバッファの末尾に改行コードを追加し、さらにその後にバッファの内容を付加する。

root ： で、 g とか G とかは、この逆だ。

g
　コピーバッファの内容を、バッファに写す。
G
　バッファの末尾に改行コードを追加し、さらにその後にコピーバッファの内容を付加する。

root ： h は hold、 g は getっていう風に覚えておくと、覚えやすいよ。

yasuoka ： はい。それから 4 行目は $pっと。これは入力の最終行だけ p を実行するってことですね。

root ： そう、出力は基本的に p でおこなうんだ。

p
　バッファの内容に改行コードを付加して、標準出力に出力する。

root ： 最後の d だけど、これが入力の次の行を読み込む命令だよ。

d
　入力の次の行をバッファに読み込み、スクリプトの最初にジャンプする。ただし入力に次の行がないときには、実行を終了する。

yasuoka ： 1 つ 1 つの命令はわかったんですけど、全体がどう動くのかわかりません。

root ： そうか。じゃ、入力として

```
/home/yasuoka/bin% ls （ぽこ）
bow     dow     hb      tac     users   where   who
/home/yasuoka/bin% ls -1 ??? （ぽこ）
bow
dow
tac
who
/home/yasuoka/bin%
```

この ls -1 ??? の出力が入った時の tac の動作を

```
/home/yasuoka/bin% ls -1 ??? | tac （ぽこ）
who
tac
dow
bow
/home/yasuoka/bin%
```

順を追って説明しようか。

yasuoka ： はい。

root ： まず 1 行目の「bow」がバッファに読み込まれた状態で、実行開始。あっと、行の末尾の改行コードはバッファには読み込まれないから、注意するようにね。

yasuoka : はい。

root : 次に h が実行されるからバッファの内容がコピーバッファに写される。つまりコピーバッファの内容も「bow」になる。それから d が実行されるんで、コピーバッファは「bow」のまま、バッファには次の行の「dow」が読み込まれる。

yasuoka : で、スクリプトの最初に戻るんですね。

root : そう。入力は 2 行目になってるから、今度は G が実行される。コピーバッファの「bow」が改行コード付きでバッファに付加されるから、バッファは「dow改bow」となる。改ってのは、改行コードね。

yasuoka : ははーん。

root : h でバッファの「dow改bow」をコピーバッファに写してから d。バッファに「tac」が読み込まれる。次に G でバッファが「tac改dow改bow」になり、 h でコピー、 d で最終行の「who」が読み込まれる。

yasuoka : いよいよ佳境ですね。

root : G で「who改tac改dow改bow」になったバッファを h でコピーした後、最終行だから p が実行されて末尾に改行コードを付けて出力。これで見事

```
who
tac
dow
bow
```

が表示されるわけだ。

yasuoka : やった。でも、その h は無駄ですね。

root : まあね。で、最後の d が実行されるけど、もう入力行はないから実行終了。めでたしめでたし。

yasuoka : めでたしめでたし。うーん、たった 5 行でこんなことができるなんて、すごーい。

root : すごいだろう。でも実はこのスクリプト、もう 1 行減らせるんだよ。

yasuoka : ええ？

```
#! /bin/sed -f
1!G
h
$!d
```

yasuoka : p がないじゃないですか。どうやって出力するんですか？

root : 実は sed のスクリプトの最後には、自動的に p と d が付加されるんだ。だから、このスクリプトは実際には

```
#! /bin/sed -f
1!G
h
$!d
p
d
```

と同じ動作をする。

yasuoka : そうなんですか。

root : これだと入力の 1 行目では h d、2 行目から最終行の 1 行前までは G h d、最終行では G h p d だから、さっきの tac と同じだろ。

yasuoka : ええ。ほんとにすごいですね。

root : sed の奥の深さは、まだまだこんなもんじゃないよ。試しに次のスクリプトを、~/bin の中に作ってみてくれるかい？ ファイル名は delcom だ。

```
#! /bin/sed -f
/^#/d
/^ *$/d
```

（間）

yasuoka : chmod してっと。

```
/home/yasuoka/bin% chmod 755 delcom（ぽこ）
/home/yasuoka/bin% rehash（ぽこ）
/home/yasuoka/bin% 
```

できました。

root : よし、ちょっとごめんね。何かいい例はないかな。

```
/home/yasuoka/bin% cat bow（ぽこ）
#! /bin/csh
# "bow" Version 2.0

if ($#argv) then
  set TIME=$argv[1]
else
```

```
  set TIME=3
endif

while ($TIME > 0)
  echo BOWWOW
  @ TIME--
end

exit (0)
/home/yasuoka/bin% 
```

お、これいいな。 delcom bow を実行してみてごらん。

yasuoka : はい。

```
/home/yasuoka/bin% delcom bow (ぽこ)
if ($#argv) then
  set TIME=$argv[1]
else
  set TIME=3
endif
while ($TIME > 0)
  echo BOWWOW
  @ TIME--
end
exit (0)
/home/yasuoka/bin% 
```

ん？ どうなったんですか？

root : # から始まる行と空行とが、削除されたんだよ。

yasuoka : あ、そうですね。

```
/home/yasuoka/bin% cat delcom (ぽこ)
#! /bin/sed -f
/^#/d
/^ *$/d
/home/yasuoka/bin% 
```

どういう仕掛けなんですか？

root : 2 行目と 3 行目の d の前に実行条件が書いてあるだろ。

yasuoka : /^#/ とかいうやつですか？

root : そうそう。その / と / に挟まれてるのが、正規表現っていうものだよ。/^#/ ってのは、バッファの先頭の 1 文字目が # だったら、真になる。ちょっと正規表現について説明しておこうか。

^	バッファの先頭
$	バッファの末尾
.	任意の 1 文字（改行コードも 1 文字とみなす）
[複数の文字]	複数の文字のいずれか 1 文字、ただし - は省略を意味する
[^複数の文字]	複数の文字以外の 1 文字、上と同じ省略形が許される
*	直前の 1 文字の 0 回以上の繰り返し
\(正規表現\)	正規表現にマッチした文字列を順に \1 … \9 に格納
\1 … \9	格納された文字列
\n	改行コード
\^	^
\$	$
\.	.
\[	[
\]	]
*	*
\/	/
\\	\
その他の文字	その文字自身

yasuoka : すると delcom の 2 行目は、バッファの 1 文字目が # だったら d を実行する、って意味ですか？

root : そう。

yasuoka : 3 行目は？

root : バッファが空っぽか空白ばっかりだったなら d を実行する、っていう意味だよ。 /^ *$/ ってのは、まず ^ がバッファの先頭、次の空白の後に * がついてるのが空白の 0 回以上の繰り返し、最後の $ がバッファの末尾だからね。つまり、バッファ全体が 0 個以上の空白で埋まっていたなら、 d を実行するってことだ。

yasuoka : だいたいわかりました。

root : で、さっきも言ったように、 sed のスクリプトの最後には p と d が付加されるから、この delcom は

```
#! /bin/sed -f
/^#/d
/^ *$/d
p
d
```

と同じ意味になる。

yasuoka : あ、そうか。# から始まる行や空行は p を実行しないから、出力されないんですね。でもどうして /^#/d なんですか？

root : て、言うと？

yasuoka : /#/d じゃダメなのかなぁ、と思って。

root : じゃあ、試しにやってごらん。

yasuoka : はい。

```
/home/yasuoka/bin% cp delcom delcom~ (ぽこ)
/home/yasuoka/bin% 
```

(間)

yasuoka : できました。

```
/home/yasuoka/bin% cat delcom (ぽこ)
#! /bin/sed -f
/#/d
/^ *$/d
/home/yasuoka/bin% delcom bow (ぽこ)
  set TIME=$argv[1]
else
  set TIME=3
endif
while ($TIME > 0)
  echo BOWWOW
  @ TIME--
end
exit (0)
/home/yasuoka/bin% 
```

あれ、 if ($#argv) then が消えちゃった。

root : /#/dってのは、バッファが # を含んでいるなら d を実行する、って意味になるからね。

yasuoka : 正規表現ってのは、バッファ全体を探してくれるんですか？

root : ^ とか $ とかの指定が無い時はね。

yasuoka : へーえ、そうか。

```
/home/yasuoka/bin% mv delcom~ delcom (ぽこ)
/home/yasuoka/bin% 
```

それから /^ *$/d ですけど、タブだけの行は削除されないんですか？

root : このままじゃダメだね。こんな風にしたらどうかな。

```
#! /bin/sed -f
/^#/d
/^[     ]*$/d
```

root : [ ] の中にたくさん空白が入ってるように見えるけど、これは空白とタブを 1 つずつ入れてるんだよ。

yasuoka : これでどうなるんですか？

root : 空白かタブかどちらかの、 0 回以上の繰り返しでバッファが埋まっていたなら d。

yasuoka : 空白とタブが交互に出てくるような行は？

root : d が実行される。

yasuoka : すると例えば /^[abc]*$/ってのは、バッファの内容が「abbcab」の時にはどうなるんですか？

root : 真になるよ。 a か b か c かの 0 回以上の繰り返しだからね。あっとそれから /^[abc]*$/ は /^[a-c]*$/ でもかまわない。

yasuoka : ワイルドカードみたいですね。

root : うん、まあ似てるけど、 * とか . とかが違うから、はっきり分けて覚えておいた方がいい。

yasuoka : はい。

root : sed には正規表現を使ったすごく強力な命令があるんだけど、今日はもうこんな時間だから、それはまた今度にしてくれるかい？

yasuoka : 強力って？

root : それはまた今度。とにかく強力なんだ。

yasuoka : はい。どうもありがとうございました。